EECS 211
Advanced System Software
Winter 2006

Assignment 3

Posted: February 2, 2006
Due: February 9, 2006

Topic: Scheduling in Nachos

Instructions:

The goal of this third assignment is to develop, implement and test task scheduling in the Nachos system. This assignment continues the previous assignment based on the "Nachos Assignment 1" described in the file **`doc/thread.ps`** of the Nachos installation. Again, the instructions below assume that you read **`doc/threads.ps`** in parallel.

Task 1: Implement a priority-based scheduler

See item 8 in **`doc/threads.ps`**.
Again, we will work in the **`threads`** directory. As you have noticed in the previous assignment, the given Nachos scheduler implements a straightforward FIFO scheduling policy. We will change that now into a priority-based policy.
With each thread, we will associate a (static) priority between 0 and 9, 0 being the highest priority (first choice).
Follow the instructions to item 8 to implement the priority scheduling. You will need to modify the code in the files **`thread.cc`**, **`thread.h`**, **`scheduler.cc`**, and **`scheduler.h`**. Note that there is not much new code to write.

Deliverable 1: (20 points)

Adjusted source files **`thread.cc`**, **`thread.h`**, **`scheduler.cc`**, and **`scheduler.h`** (with proper comments!).

Task 2: Test the priority-based scheduler

In order to test your code, we will reuse the bounded-buffer scenario from the previous assignment. This time, however, construct a system of 3 producers, P1, P2, P3, and 1 consumer C, which will communicate over 1 bounded buffer with a capacity of 5 characters. Producer P1 should send the string "AAA", P2 should send "BBB", and P3 "CCC". The consumer thread simply should print the string of the received characters.

Depending on the priorities assigned to the 4 threads, the string received by the consumer will be different. The usage of the buffer (number of characters in it) also will depend on the priorities assigned to the threads.

Find priorities for the four threads, such that the

(a) output is “AAABBBCCC” with minimal buffer usage
(b) output is “CCCBBBAAA” with maximal buffer usage

These conditions should hold even in the case of preemptive scheduling!
You may want to use the **`-rs <seed>`** option to enable random context switches to simulate a preemptive scheduler.

Deliverable 2: (15 points)

a) Modified source code **`threadtest.cc`** with the bounded buffer scenario as described above
b) A text file **`priority.txt`** explaining the necessary priorities of the threads for each of the two cases (a) and (b)
c) A type-script **`priority.log`** (copy of your shell outputs) showing that your program correctly produces the expected output string

Task 3: Cooperative Multithreading (Extra Credit!)

Can you come up with a scheme (and working implementation) that lets the consumer receive the string “ABCABCABC”? The three producer threads still should send the strings “AAA”, “BBB”, and “CCC”, and the buffer should be unmodified.

Hint: Consider the Thread->Yield() function to make the producers “cooperative”.

Optional Deliverable 3: (10 extra points)

d) Modified source code **`extra_threadtest.cc`**
e) A text file **`extra.txt`** explaining your implementation
f) A type-script **`extra.log`** (copy of your shell outputs) showing that your program correctly produces the expected output string

Submission instructions:

To submit your homework, send an email with subject “EECS 211 HW 3” to the course instructor at doemer@uci.edu. Please submit the deliverables listed above as attachments.

To ensure proper credit, be sure to send your email before the deadline: February 9, 2006, 11:59pm.

--
Rainer Doemer (ET 444C, x4-9007, doemer@uci.edu)